Issues in Developing Object-Oriented Database Systems for Real-Time Applications[1]

Juhnyoung Lee and Sang H. Son

Department of Computer Science
University of Virginia
Charlottesville, VA 22903

Myung-Joon Lee[2]

Department of Computer Science
University of Ulsan
Ulsan, Kyung-Nam 680-749, Korea

Abstract

Database systems for real-time applications must satisfy timing constraints associated with transactions, in addition to maintaining data consistency. Recently, interests in object-oriented databases have been growing for non-traditional applications of database systems, and several real-time applications are being developed using an object-oriented paradigm. The object-oriented approach seems promising for developing complex real-time database applications. However, it is not clear whether object-oriented database systems would be superior than relational database systems for supporting real-time applications. In this paper, we address issues that must be investigated in order to design and develop an object-oriented database system for real-time applications. Also, we present a model that integrates features for scheduling real-time transactions with the traditional object-oriented database model.

1. Introduction

A *real-time database system* (RTDBS) is a transaction processing system where transactions have explicit timing constraints. Typically, a timing constraint is expressed in the form of a deadline, a certain time in the future by which a transaction needs to be completed. A deadline is said to be *hard* if it cannot be missed or else the result is useless. If a deadline can be missed, it is a *soft* deadline. With soft deadlines, the usefulness of a result may decrease after the deadline is missed. In RTDBS, the correctness of transaction processing depends not only on maintaining consistency constraints and producing correct results, but also on the time at which a transaction is completed. Transactions must be scheduled and processed in such a way that they can be completed before their corresponding deadlines expire. Real-time database systems are being used for a variety of applications such as process control, mission critical applications in command and control systems and radar systems, computer integrated manufacturing systems, and air traffic control systems, among others.

Conventional data models and databases are not adequate for time-critical applications, since they are not designed to provide features required to support real-time transactions. They are designed to provide good average performance, while possibly yielding unacceptable worst-case response times. Very few of them allow users to specify or ensure timing constraints. During the last few years, several research and development efforts on RTDBSs have been reported [19, 21, 22, 26]. However, almost all of them are based on relational data model. Although object-oriented database systems have received a lot of attention for last several years, not much work has been done in investigating how object-oriented model can benefit database systems for real-time applications. Only recently, object-oriented data models have attracted the attention of researchers in RTDBSs [5, 6, 13, 14, 15].

2. Preliminary Questions

There are several questions to be answered, before object-oriented database can be considered for real-time applications. First, do we need object-oriented data models to satisfy real-time database requirements? Related questions are: Are the features of object-oriented data models helpful/necessary to satisfy timing constraints? Or do they interfere with timely execution of transactions? Why do we need to consider object-oriented models for real-time database systems? In general, using object-oriented data models for an RTDBS does not directly help the system to improve timeliness or to guarantee deterministic behavior of transaction execution, because none of the object-oriented data model's features provides active pursuit of timely/deterministic processing of transactions. In addition, poor performance of current

1. This work was supported in part by ONR, IBM and CIT.
2. Currently visiting Department of Computer Science at University of Virginia.

0-8186-6375-8/94 $04.00 © 1994 IEEE

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1994		**00-00-1994 to 00-00-1994**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Issues in Developing Object-Oriented Database Systems for Real-Time Applications	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Virginia,Department of Computer Science,151 Engineer's Way,Cahrlottesville,VA,22094-4740**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**5**	

object-oriented database systems partly due to the lack of efficient implementation techniques will have a negative impact on satisfying timing constraints. Two major benefits of object-oriented data models are (1) better support of advanced data-intensive applications by providing the capabilities for modeling, storing and manipulating *complex objects*, and (2) better software engineering in building large and complex application systems providing support for *encapsulated objects*. The need for supporting real-time database requirements with object-oriented data models may arise because real-time applications may require modeling complex encapsulated objects.

Is there any inherent problem in object-oriented data models in satisfying timing constraints of real-time applications? It is obvious that the basic features of object-oriented data models (objects, attributes, methods, messages, classes, class hierarchy, and inheritance) do not directly help timely processing of transactions. It is also true, however, that none of them particularly interferes with active pursuit of timely processing of transactions, except for the potential lack of efficient implementation techniques. Thus, issues in supporting real-time requirements with an object-oriented data model lie in extending the object-oriented data model to include specification of timing constraints on objects (more specifically on attributes/methods), and to actively pursue timely processing of transactions, rather than in combating any incompatibility between object-oriented data models and real-time requirements.

3. An Extended Object-Oriented Database Model

In this section, we first describe the traditional model of object-oriented databases [3, 8]. Since the notion of nesting is natural in object-oriented databases [1, 8, 20], the model allows objects to include methods that are not necessarily atomic and may invoke other methods. Then, we briefly discuss features required for scheduling transactions with real-time requirements, and extend the model of object-oriented databases to include policies for scheduling real-time transactions.

An *object-oriented database* is a collection of *classes* and *instances* of these classes. Both classes and instances are referred to as *objects*. A class defines a set of attributes for its instances and procedures through which instances can be manipulated. The procedures associated with a class are referred to as *methods*, and a method may invoke other methods on other objects in the database. In this model, we allow *inheritance* of properties (attributes and methods) between classes, i.e., the class is structured as a hierarchy. All subclasses of a class inherit all properties defined for the class and have additional properties local to the subclass.

Users of an object-oriented database access the instance objects by executing methods. Since multiple users often may need to access several classes and instances, the traditional transaction model for database systems can be used to ensure atomicity of user interactions. Users access the database by executing transactions, where a *transaction* is a partially ordered set of operations on class and instance objects. We use *commutativity* as the basis for determining whether a particular operation invocation can be allowed to execute concurrently with those in progress [24]. Two operations *commute* if the order in which they execute does not affect the results of the operations, i.e., results returned by the operation as well as the resulting state of the objects accessed. Two operations in different transactions *conflict* with each other if they do not commute.

To include nested transactions in this object model, we assume that a method execution is a transaction which may invoke atomic operations or invoke other methods on other objects. Namely, an operation in a transaction may be an atomic operation or another transaction, and now the transaction imposes a tree structure. Hadzilacos and Hadzilacos [8] established an analogue to the classical serializability theorem to prove the correctness of nested transaction execution in object-oriented databases. As in the classical serializability theorem, the correctness of a history H over a set of nested transactions can be determined by constructing the serialization graph of H denoted as $SG(H)$. $SG(H)$ is a directed graph whose nodes correspond to the transactions in H and whose edges capture orderings of transactions that must be obeyed by an equivalent serial history. In [8], it was shown that if $SG(H)$ is acyclic then H is serializable on the basis of view serializability. To synchronize the concurrent execution of nested transactions, a concurrency control protocol, namely *nested two-phase locking* (N2PL) has been proposed [17]. In [8], the protocol has been modified to synchronize nested transactions in object-oriented databases, and its correctness was argued by using the notion of serialization graphs for nested transactions. We use this protocol as the basic synchronization mechanism for transaction execution in object-oriented databases.

In order for an object-oriented database to support real-time applications, we need to integrate features for scheduling real-time transactions with the conventional object-oriented database model. There are four major policies for scheduling transactions in real-time requirements: (1) *priority assignment*, i.e., a policy for assigning priorities to transactions, (2) *eligibility test*, i.e., a policy to determine which transactions are eligible for service, (3) *concurrency control*, i.e., a basic synchronization mechanism and (4) *conflict-resolution*, i.e., a policy for resolving conflicts between two (or more) transactions that access the same data object. Each

scheduling policy should work cooperatively to maximize the number of transactions that meet their deadlines.

Transaction scheduling in an RTDBS can be studied from several different perspectives. This largely depends on how the system is specified in terms of data consistency requirements and timing constraints. In this study, we assume that data consistency is defined by the correctness notion of *serializability* (i.e., a relaxation of serializability is not considered for improving timeliness.), and that the timing constraints associated with transactions are *firm deadlines* (i.e., transactions which miss their deadlines are useless and need to be discarded from the system.). In addition, we assume that transactions arrive sporadically with unpredictable arrival times, and that the data and resource requirements of each transaction is unknown to concurrency control beforehand.

Our model of an object-oriented database for scheduling real-time transactions is an extension of the traditional the object-oriented database model to include the four features for scheduling real-time transactions. Considering the assumptions we made regarding real-time requirements for transactions, we choose to use the following strategies for each of the four scheduling policies. First, to assign priorities to transactions, we use the *Earliest Deadline First* (EDF) algorithm. Since the real-time scheduling theory ensures that the EDF algorithm, which assigns the highest priority to the transaction with the earliest deadline, is optimal for dynamic priority assignment [12], EDF is a plausible choice for the given transaction model.

Second, to maintain data consistency, we employ the *nested two-phase locking* protocol. As mentioned in the previous section, the correctness of the protocol has been proven for the execution of nested transactions in object-oriented databases.

Third, for conflict resolution, we employ the *high priority* and *wait promote* [2] schemes to be incorporated into the basic concurrency control mechanism. i.e., N2PL. We also consider a conditional conflict resolution scheme discussed in [9], which switches between the two schemes using the information of the lock holding transaction's current state.

Finally, we abort transactions that have missed their deadlines by using eligibility test. Due to the firm deadline assumption, the aborted transactions are discarded from the system. One salient point about the eligibility test used in this model is that it can screen out transactions that not only have missed but also are about to miss their deadlines. To decide for the eligibility of transactions, we use the *minimal execution times* of methods defined on objects. The minimal execution times of methods may be relatively easy to compute by empirically measuring their running times under no contention for objects among transactions. Note that the nested structure of transaction execution helps the computation and use of the minimal execution times of component methods in a nested transaction. The details of this extended model of object-oriented databases for scheduling real-time transactions and related concurrency control protocols are given in [13].

4. Concurrency Control Issues

In this section, first we consider the difficulties in synchronizing concurrent execution of transactions in object-oriented databases and discuss research directions to enhance the performance of concurrency control in object-oriented databases. Then we discuss a simple object-oriented database system model for the development of a complete object-oriented database for real-time applications.

Object-oriented databases generalize the traditional database model in several ways. First, nested executions of transactions on objects are natural since a method may invoke another method on some other objects. Second, instead of simple read and write operations on database objects, object-oriented databases permit arbitrary operations on objects. Finally, inheritance in object-oriented databases allows class hierarchies. These properties often make the problem of ensuring data consistency in object-oriented databases more difficult, because objects of arbitrary complexity become the unit of locking (and thus less concurrency in transaction execution is resulted), and sometimes concurrency control requires to lock not only the object accessed by a transaction, but also several other objects not directly accessed by the transaction. Specifically, due to inheritance, (1) while a transaction accesses instances of a class, another transaction should not able to modify the definition of any of the super classes of the class, and (2) while a transaction is evaluating a query, a set of class sub-hierarchies must not be modified by a conflicting transaction [11].

In order to overcome the inefficiency in ensuring data consistency in object-oriented databases, an extensive study on improving concurrency in transaction execution in objected-oriented databases has been done. Three major approaches are: (1) exploiting the structure of complex objects for enhanced concurrency or reduced overhead, (2) exploiting the semantics of operations on encapsulated objects to enhance concurrency, and (3) automating the process of extracting possible concurrency from the specification of objects.

Examples of approach (1) include the concurrency control mechanisms of Orion [7] and O_2 systems [4]. Orion uses locking on three orthogonal types of hierarchy: granularity locking (to minimize the number of locks to be

set), class-lattice locking (to handle class hierarchy), and composite object locking (to handle object clusters) [7]. The eight lock modes used in Orion only consider read and write operations, and require a complex lock compatibility table without considering operation semantics. Approach (2) is related to work on concurrency control for *abstract data types* (ADTs), and the use of fine and *ad hoc* commutativity relation of operations in such ADTs as sets, maps, stacks, and counters [10, 23, 24]. Examples of previous work applying this approach for concurrency control in object-oriented databases include [1, 18, 20]. Finally, an example of approach (3) can be found in [16]. The degree of concurrency that can be extracted from this static analysis of operation specification at the stage of compilers seems limited.

Difficulties in managing transactions caused by objects of arbitrary complexity and their hierarchical relationship also make the implementation of an object-oriented database system complicated, and have an adverse impact on its capability to support real-time applications. In order for an object-oriented database system to efficiently support real-time applications, the system needs to be carefully designed to mitigate the complexity in transaction management. Now we describe a simple object-oriented database system model that is designed taking this consideration into account.

Two key concepts of this model are *atomic objects* and *class manager*. Atomic objects are basic entities for ensuring atomicity of transactions in this model, and the class manager is the major vehicle that lessens the complexity involved in transaction management in the object-oriented database system. The notion of atomic objects was studied in a number of papers, including [10, 23, 24, 25], and was first used in the context of real-time object-oriented databases in [5]. Atomic objects are ones that provide appropriate synchronization and recovery. Encapsulating the synchronization and recovery needed to support atomicity in the implementations of the shared objects is feasible because methods defined in an object provide the only means to access the object's data, and data contention can occur only among method invocations within the object. With atomic objects, we can enhance modularity; in addition, we can increase concurrency among transactions by using information about the specifications of the shared objects.

For an efficient support of transaction management in an object-oriented database system, we believe that the task of managing class hierarchies and method commutativities should be performed by a single module. Thus, our model uses a class manager to maintain the definition of classes, and the information of class hierarchies due to inheritance and composition. The class manager uses this information to maintain commutativity relation among methods of each class, and provides concurrency control with its information when requested. Note that in this model, the commutativity of method invocations is statically determined and maintained by the class manager, while concurrency control which uses the information of method commutativity, is dynamically performed by each atomic object.

In [5, 6], DiPippo and Wolfe proposed a comprehensive model for real-time object-oriented databases and flexible approaches to processing real-time transactions in such a model. To determine compatibility relation of methods, their approach considers not only a broad domain of semantic information affecting logical consistency, but also temporal consistency constraints. In addition, the approach allows a wide range of correctness criteria for logical consistency that relax serializability. Our work (described in this paper and [13]) is different from theirs in a number of aspects. First, we use the correctness notion of serializability to define data consistency requirements, but do not consider any relaxation of serializability for improving timeliness. The work in [5, 6] proposed a concurrency control technique that allows imprecision to accumulate in data values and in transactions as a result of trading off logical consistency and temporal consistency.

Second, in our system, commutativity of methods (and method invocations in case of the range of parameter values being discrete) is determined a priori at compile-time, and run-time checking of commutativity is as efficient as for compatibility. We believe that the simple and efficient run-time checking is beneficial to support real-time transactions, because that may increase predictability in transaction execution. One drawback of this scheme is that concurrency level may be relatively limited, because it does not exploit dynamic information about objects. In [5, 6], granting of locks is controlled by run-time evaluation of a set of preconditions and compatibility functions defined on every ordered pair of methods. This approach seems to increase concurrency level among transactions at the cost of run-time overhead.

Finally, in [13] we proposed a synchronization mechanism for scheduling real-time transactions in an object-oriented database, which uses the minimal execution time estimates of methods to decide eligibility of transactions for service. Note that our system model helps to accurately estimate the minimal execution times due to its run-time efficiency. The minimal execution times of methods are useful in scheduling real-time transactions in an object-oriented database, while in general the worst execution time estimates may not be helpful due to large variance of transaction execution time in typical database systems.

5. Conclusion

In summary, object-oriented data models do not directly help the database systems to improve timeliness or to guarantee deterministic behavior of real-time applications. They do not provide features to support active pursuit of timely and deterministic processing of transactions. However, since object-oriented database models allow better support for managing complex objects and encapsulation, real-time systems that need to handle large and complex applications would require an object-oriented approach. Considering the implications of inherent complexity of concurrency control in object-oriented paradigms, we need to start from a simple model that can be easily extended to support real-time transactions and temporal constraints of real-time data. The model outlined in this paper can be one candidate for the development of a complete object-oriented database for real-time applications.

References

[1] Agrawal, D., and A. E. Abbadi, "A Non-Restrictive Concurrency Control for Object Oriented Databases," *Proc. of the 3rd Int. Conf. on Extending Data Base Technology*, Vienna, Austria, March 1992.

[2] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Trans. on Database Systems*, 17(3):513-560, Sept. 1992.

[3] Banerjee, J., W. Kim, H. J. Kim, and H. F. Korth, "Semantics and Implementation of Schema Evolution in Object-Oriented Databases," *ACM SIGMOD International Conference on Management of Data*, May 1987.

[4] Cart, M., and J. Ferrie, "Integrating Concurrency Control into an Object-Oriented Database System," *Proc. of the Int. Conf. on Extending Data Base Technology*, 1990.

[5] DiPippo, L. B. C., and V. F. Wolfe, "Object-Based Semantic Real-Time Concurrency Control," *Proc. of the 14th IEEE Real-Time System Symposium*, Dec. 1993.

[6] DiPippo, L. B. C., and V. F. Wolfe, "Distributed Object-Based Semantic Real-Time Concurrency Control with bounded imprecision," *Technical Report*, TR93-227, Department of Computer Science & Statistics, University of Rhode Island, April 1994.

[7] Garza, J. F., and W. Kim, "Transaction Management in an Object-Oriented Database System," *ACM SIGMOD International Conference on Management of Data*, June 1988.

[8] Hadzilacos, T. and V. Hadzilacos, "Transaction Synchronization in Object Bases," *Journal of Computer and System Sciences*, 43(1):2-24, August 1991.

[9] Huang, J., J. A. Stankovic, K. Ramamritham, and D. Towsley, "On Using Priority Inheritance in Real-Time Databases," *Proc. of the 12th IEEE Real-Time System Symposium*, December 1991.

[10] Herlihy, M. P., and W. E. Weihl, "Hybrid Concurrency Control for Abstract Data Types," *ACM Symposium on Principles of Database Systems*, Austin, Texas, March 1988, pp. 201-210.

[11] Kim, W., "Object-Oriented Databases: Definition and Research Directions," *IEEE Transactions on Knowledge and Data Engineering*, 2(3), September 1990.

[12] Liu, C. L. and J. W. Layland, "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment," *Journal of ACM*, 20(1), January 1973.

[13] Lee, J., M. J. Lee and S. H. Son, "Scheduling Real-Time Transactions in Object-Oriented Databases," *Technical Report*, Department of Computer Science, University of Virginia, April 1994.

[14] Lortz, V. B., I. P. Mangiavacchi, and K. G. Shin, "An Object-Oriented Approach to Integrating Real-Time Manufacturing Systems," *Technical Report*, Dept. of Electrical Engineering and Computer Science, Univ. of Michigan, 1993.

[15] Lee, J., and S. H. Son, "Semantic-Based Concurrency Control for Object-Oriented Database Systems Supporting Real-Time Applications," *Proc. of the 6th Euromicro Workshop on Real-Time Systems*, June 1994.

[16] Malta, C., J. Martinez, "Automating Fine Concurrency Control in Object-Oriented Databases," *Proc. of the 9th Int. Conf. on Data Engineering*, Vienna, Austria, April 1993.

[17] Moss, J. E. B., *Nested Transactions: An Approach to Reliable Distributed Computing*, MIT Press, Cambridge, Massachusetts, 1985.

[18] Muth, P., T. C. Rakow, G. Weikum, P. Brossler, C. Hasse, "Semantic Concurrency Control in Object-Oriented Database Systems," *Proc. of the 9th Int. Conf. on Data Engineering*, Vienna, Austria, April 19-23, 1993.

[19] Ramamritham, K., "Real-Time Databases," *International Journal of Distributed and Parallel Databases*, Vol. 1, No. 1, 1992.

[20] Rakow, T. C., J. Gu, E. J. Neuhold, "Serializability in Object-Oriented Database Systems," *Proc. of the 6th International Conference on Data Engineering*, April 1990.

[21] Special Issue on Real-Time Database Systems, *ACM SIGMOD Record*, March 1988.

[22] Son, S. H., "Real-Time Database Systems: A New Challenge," *Data Engineering*, 13(4): 51-57, Dec. 1990.

[23] Schwartz, P. M., and A. Z. Spector, "Synchronizing Shared Abstract Types," *ACM Transactions on Computer Systems*, 2(3):223-250, 1984.

[24] Weihl, W., "Commutativity-Based Concurrency Control for Abstract Data Types," *IEEE Transactions on Computers*, 37(12):1488-1505, December 1988.

[25] Weihl, W., "Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types," *ACM Transactions on Programming Languages and Systems*, 11(2):249-282, April 1989.

[26] Yu, P. S., K.-L. Wu, K.-J. Lin, and S. H. Son, "On Real-Time Databases: Concurrency Control and Scheduling," *Proceedings of the IEEE*, 82(1):140-157, January 1994.